Intro

Graphs are used to help solve problems such as

- Finding the shortest path between two points
- Scheduling (e.g., course and room example)
- *Seven Bridges of Konisgsberg*

Terminology

- A **Graph** G is a set of **vertices** V and a set of **edges** E (which are pairs of vertices).
- Two vertices are **adjacent** if they are joined by an edge.
- Graphs can be **directed** or **undirected**.
  - ⇒ An undirected graph does not differentiate between edge (u,v) and (v,u)
  - ⇒ A directed graph may have an edge going from vertex u to vertex v, but not from vertex v to u.

- An edge is **incident** to a vertex, if the vertex is one of the edges **endpoints**.
- The **degree** of a vertex, denoted deg(v), is the number of edges that are incident with v.
- In a directed graph, a vertex v can have an **in-degree**, which is the number of incoming edges to v, and an **out-degree**, which is the number of outgoing edges from v.
- A **path** is the set of vertices and edges that can be followed to go from one particular vertex to another. A **cycle** is a path that starts and ends at the same vertex. (A tree is a graph with no cycles (i.e. it is **acyclic**))

- A graph is **connected** if for any 2 vertices, there is a path between them.
- A **subgraph** of graph G is a graph H whose vertices and edges are subsets of the vertices and edges of G.
- The number of **components** of a graph G, refers to the maximal connected subgraphs of G.

Graph Properties

Let m be the number of edges in simple graph G, and let n be the number of vertices in G.

(Note: A simple graph means a graph with no self-loops or multi-edges)

Proposition 1

If G is undirected then,

$$\sum_{v \in G} deg(v) = 2m$$

proof: We count every edge (u,v) twice, once when we sum deg(u) and once when we sum deg(v).

Proposition 2

If G is directed then,

$$\sum_{v \in G} indeg(v) = \sum_{v \in G} outdeg(v) = m$$

proof: Each edge (u,v) gets counted once for the out-degree and once for the in-degree.

Proposition 3

If G is undirected,

$$m \leq n(n-1)/2$$

proof: The maximum degree of each vertex can be at most (n-1). So by the first proposition, $2m \leq n(n-1)$.

Graph ADT's

Some of the graph methods/procedures:

- Getting global information about a graph:
  - $\Rightarrow$ numVertices() : Returns the number of vertices in G
  - $\Rightarrow$ numEdges() : Returns the number of edges in G
  - $\Rightarrow$ vertices() : Returns all the vertices in G
  - $\Rightarrow$ edges() : Returns all the edges in G

- For Undirected Graphs:
  - $\Rightarrow$ degree(v) : Returns the degree of v
  - $\Rightarrow$ incidentEdges(v) : Returns the edges incident to v
  - $\Rightarrow$ adjacentVertices(v) : Returns the vertices adjacent to v
  - $\Rightarrow$ areAdjacent(v,w) : Returns true is vertices are adjacent

- For Directed Graphs:
  - $\Rightarrow$ inDegree(v): Returns the in-degree of v
  - $\Rightarrow$ outDegree(v): Returns the out-degree of v
  - $\Rightarrow$ inIncidentEdges(v): Returns all the incoming edges to v
  - $\Rightarrow$ outIncidentEdges(v): Returns all the outgoing edges from v
  - $\Rightarrow$ inAdjacentVertices(v): Returns all the vertices adjacent to v along incoming edges to v
  - $\Rightarrow$ outAdjacentVertices(v): Returns all the vertices adjacent to v along outgoing edges from v

- For Updating Graphs:
  - ⇒ insertEdge(v,w,o): Insert an undirected edge between v and w
  - ⇒ insertDirectedEdge(v,w,o): Insert a directed edge between v and w
  - ⇒ insertVertex(o): Insert a new (isolated) vertex
  - ⇒ removeVertex(v): Remove a vertex and all its incident edges
  - ⇒ removeEdge(e): Remove an edge
- + others

Data Structures for Graphs

- Edge List
- Adjacency List
- Adjacency Matrix

Edge List

Easy to get edge-based information (eg. what vertices are endpoints for a particular edge) Harder to get vertex-based information.

Run Times

numVertices(), numEdges() : O(1)

vertices() : O(n)

edges() : O(m)

degree() : O(1)

incidentEdges(v) : O(m)

adjacentVertices(v) : O(m)

areAdjacent(v,w) : O(m)

Adjacency List

Extends edge list, allowing easier access to vertex-based information.

Run Times

numVertices(), numEdges() : O(1)

vertices() : O(n)

edges() : O(m)

degree() : O(1)

incidentEdges(v) : O(deg(v))

adjacentVertices(v) : O(deg(v))

areAdjacent(v,w) : O(min(deg(v), deg(w))

Adjacency Matrix

Able to determine if two vertices are adjacent in constant time. However, it takes up a lot of space.

Run Times

numVertices(), numEdges() : O(1)

vertices() : O(n)

edges() : O(m)

degree() : O(1)

incidentEdges(v) : O(n)

adjacentVertices(v) : O(n)

areAdjacent(v,w) : O(1)